

Subset Sum Problems With Digraph Constraints

Laurent Gourvès¹, Jérôme Monnot¹, and Lydia Tlilane¹

Université Paris-Dauphine, PSL Research University, CNRS, LAMSADE, 75016 Paris, France
 {laurent.gourves, jerome.monnot, lydia.tlilane}@dauphine.fr

Abstract. We introduce and study four optimization problems that generalize the well-known subset sum problem. Given a node-weighted digraph, select a subset of vertices whose total weight does not exceed a given budget. Some additional constraints need to be satisfied. The (weak resp.) digraph constraint imposes that if (all incoming nodes of resp.) a node x belongs to the solution, then the latter comprises all its outgoing nodes (node x itself resp.). The maximality constraint ensures that a solution cannot be extended without violating the budget or the (weak) digraph constraint. We study the complexity of these problems and we present some approximation results according to the type of digraph given in input, e.g. directed acyclic graphs and oriented trees.

Key words. Subset Sum, Maximal problems, digraph constraints, complexity, directed acyclic graphs, oriented trees, PTAS.

1 Introduction

This paper deals with four optimization problems which generalize the well-known SUBSET SUM PROBLEM (SS in short). Given a digraph $G = (V, A)$ such that each $x \in V$ has a nonnegative weight $w(x)$, we search for $S \subseteq V$ satisfying some constraints. As for SS we have a *budget constraint* which imposes that $w(S) \equiv \sum_{x \in S} w(x)$ does not exceed a given bound B . We depart from SS by considering the following constraints. The *digraph constraint* imposes to insert a node in S if one of its incoming nodes in G appears in S . A weaker form, called *weak digraph constraint*, imposes to insert a node in S if *all* its incoming nodes in G appear in S . Our last constraint requires maximality with respect to the previous constraints. A set S satisfies the *maximality constraint* if there is no $S' \supset S$ satisfying the budget and the (weak resp.) digraph constraint.

Given a digraph $G = (V, A)$ and a budget B , the four problems studied in this article are the following. SUBSET SUM WITH DIGRAPH CONSTRAINTS (SSG in short) is to find S that maximizes $w(S)$ under the budget and the digraph constraints. In SUBSET SUM WITH WEAK DIGRAPH CONSTRAINTS (SSGW in short), we seek S that maximizes $w(S)$ under the budget and the weak digraph constraints. For the MAXIMAL SUBSET SUM WITH DIGRAPH CONSTRAINTS (MAXIMAL SSG in short) we search for S with minimum weight under the constraints of budget, digraph and maximality. Finally, MAXIMAL SUBSET SUM WITH WEAK DIGRAPH CONSTRAINTS (MAXIMAL SSGW in short) aims to find S with minimum weight under the constraints of budget, weak digraph and maximality. The fact that we minimize $w(S)$ will become clear from the possible applications.

Let us motivate SSG in a scheduling context (other applications are given in [23]). A processor is available during a period of length B and there is a set of tasks to be executed on it. Each task x is represented by a vertex of a graph and has a duration $w(x)$. We seek a subset of tasks whose total duration does not exceed B . The digraph provides dependency constraints between the tasks, i.e. there is an arc (i, j) if task i requires the output of task j . The goal is to maximize the utilization of the processor during the time window.

Keep this scheduling example but replace the processor by a lazy bureaucrat who has to execute some tasks. Everyday the bureaucrat is in his office for a period of length B and the set of tasks S that he selects must be executed within this period. The maximization of $w(S)$ does not reflect the wish of the lazy bureaucrat who is interested in working as little as possible. His goal is rather to minimize $w(S)$. However, $S = \emptyset$ is not a realistic solution because the worker's employer finds unacceptable to ignore a task if there is enough time to execute it. This example, taken from [2], motivates MAXIMAL SSG and its constraint of maximality.

Let us motivate SSGW with another application. The different modules of a program are represented by a digraph $G = (V, A)$ in the sense that (x, y) belongs to A whenever module y receives informations from module x . An updated version of the program is to be deployed. Updating module x induces a cost of $w(x)$ and there is a global budget of B . We want to select a subset S of modules, candidates for the update, such that $w(S) \leq B$ and if all the predecessors of a module y are updated, then y must also be updated otherwise y works in a faulty manner.

In order to justify the study of MAXIMAL SSGW, suppose the user of the program pays an external company B dollars for updating the software. If S denotes the set of updated modules, then the revenue of the company, to be maximized, is equal to $B - w(S)$. Meanwhile, the user finds S acceptable if B is exceeded with any extra update.

Our purpose is to study SSG, SSGW, MAXIMAL SSG and MAXIMAL SSGW from a theoretical viewpoint. The complexity and approximability of these problems are analyzed for various topologies of the input digraph.

To the best of our knowledge, these problems are new, except SSG which is a special case of the PARTIALLY-ORDERED KNAPSACK problem (also known as the PRECEDENCE-CONSTRAINED KNAPSACK PROBLEM and it will define later) [22,23,19]. MAXIMAL SSG and MAXIMAL SSGW generalize the LAZY BUREAUCRAT PROBLEM with common deadlines and arrivals [2,14,15,18] representing the *maximal* version of SS. We aim at extending this problem with (weak) digraph constraints on digraphs.

Our main results are: the four problems are **NP**-hard, even for simple classes of digraphs. This is true even in in-rooted and out-rooted trees. SSG is also strongly **NP**-hard in 3-regular digraphs and this result is tight according to degree parameters. MAXIMAL SSG is strongly **NP**-hard in directed acyclic graphs (DAG) with a unique sink and 3 weights. There is also a reduction preserving approximation for the four problems in DAG with maximum in-degree 2. However, some classes of graphs make the problems solvable in polynomial time or approximable within any given error. The class of oriented trees admits non-trivial dynamic programming algorithms. In tournament graphs, SSG and MAXIMAL SSG are polynomial. We also provide approximation schemes for SSG and MAXIMAL SSG in DAG.

The present paper is organized as follows. Section 2 contains some definitions on graphs that we use throughout the paper and a formal definition of our problems. Section 3 makes an overview of related works. Then, we present some complexity results for the four problems according to the topology of the digraph: regular digraphs are studied in Section 4, DAG in Section 5 and oriented trees in Section 6. Dynamic programming algorithms are provided for oriented trees in Section 6. In Section 7, we propose approximation schemes for SSG and MAXIMAL SSG in DAG. Some perspectives are given in Section 8.

2 Definitions and concepts

2.1 Graph terminology

A directed graph (or *digraph*) is a graph whose edges have a direction. Formally, a digraph G is a pair (V, A) where V and A are the *vertex set* and the *arc set*, respectively. Given two vertices x and y , the notation (x, y) means the arc that goes from x to y and $[x, y]$ is an edge (a non-oriented arc).

The *in-neighborhood* (and the *in-degree*) of a vertex v in G denoted by $N_G^-(v)$ and $\deg_G^-(v)$ respectively are defined by $N_G^-(v) = \{u \in V : (u, v) \in A\}$ and $\deg_G^-(v) = |N_G^-(v)|$. Similarly, the *out-neighborhood* and the *out-degree*, $N_G^+(v)$ and $\deg_G^+(v)$ are defined by $N_G^+(v) = \{u \in V : (v, u) \in A\}$ and $\deg_G^+(v) = |N_G^+(v)|$. A vertex with $\deg_G^-(v) = 0$ is called a *source* and similarly, a vertex with $\deg_G^+(v) = 0$ is called a *sink*. The neighborhoods of a vertex v is defined by the set $N_G(v) = N_G^-(v) \cup N_G^+(v)$ and its degree is $\deg_G(v) = \deg_G^-(v) + \deg_G^+(v)$. A graph is k -regular if the degree of each node is k .

A *directed path* $\mu_G(x, y)$ from x to y in G is a succession of vertices (v_1, \dots, v_k) where $v_1 = x$, $v_k = y$ and $(v_i, v_{i+1}) \in A$ for every $i = 1, \dots, k - 1$. A *circuit* C is a path of positive length from x to x .

Given a subset of vertices $S \subseteq V$, we denote by $G - S$ the subgraph induced by $V \setminus S$.

In this paper, we also consider some special classes of digraphs: an *acyclic digraph* (or DAG for Directed Acyclic Graph) is a digraph without circuit. It is well known that a DAG has a source and a sink. An *oriented tree* is a digraph formed by orienting the edges of an undirected tree and an *out-rooted tree* (*in-rooted tree* resp.) is an oriented tree where the out-degree (in-degree resp.) of each vertex is equal to 1. A root (anti-root resp.) is a vertex without any in-neighborhood (out-neighborhood resp.). Finally, a *tournament* is an oriented graph where the underlying graph is a complete graph, or equivalently there is exactly one arc between any two distinct vertices.

In this document, we only consider simple digraphs, i.e. with no loop and no multiple arc.

2.2 Subset Sum problems

SUBSET SUM WITH DIGRAPH CONSTRAINTS

The first problem is called SUBSET SUM WITH DIGRAPH CONSTRAINTS (SSG in short) and its input is a digraph $G = (V, A)$, a non-negative weight $w(i)$ for every node $i \in V$, and a positive bound B . The weight of $S \subseteq V$ is denoted by $w(S)$ and defined as $\sum_{i \in S} w(i)$. A feasible solution S is a subset of V satisfying the following constraints.

$$\forall x \in S, (x, y) \in A \Rightarrow y \in S \quad (1)$$

$$w(S) \leq B \quad (2)$$

Constraints (1) are called the *digraph constraints* while (2) corresponds to a *budget constraint*. Formally, the problem is defined by:

SSG
<i>Input:</i> a node weighted digraph $G = (V, A, w)$ and a bound B .
<i>Output:</i> $S \subseteq V$ satisfying (1) and (2).
<i>Objective:</i> maximize $w(S)$.

Obviously, this optimization problem is related to the exact decision version where we try to decide if there is a subset S satisfying (1) with $w(S) = B$, which is a natural generalization of the standard SUBSET SUM decision problem (see Problem [SP13], page 223 in [16]) by considering the digraph without arcs, i.e., $A = \emptyset$.

Let us introduce an intermediary decision problem, called CARDINALITY SSG in the rest of the paper. The input consists of a digraph $G = (V, A)$, a bound B , an integer $p \leq |V|$ and a weight function w on the nodes satisfying $1 \leq w(v) \leq B - p$. The problem is to decide if there exists $J \subseteq V$ such that $w(J) = B$, $|J| = p$ and J satisfies the digraph constraints (1).

MAXIMAL SUBSET SUM WITH DIGRAPH CONSTRAINTS

This new problem is called MAXIMAL SUBSET SUM WITH DIGRAPH CONSTRAINTS (MAXIMAL SSG in short) and its input is the same as for SSG. A feasible solution S is a subset of V satisfying (1), (2) and the following third constraint:

$$\text{There is no } S' \supset S \text{ such that } S' \text{ satisfies (1) and (2).} \quad (3)$$

This last condition is called the *maximality constraint* and it corresponds to the notion of maximal subset satisfying the digraph constraint. As opposed to SSG, the goal of MAXIMAL SSG is to *minimize* $w(S)$. Formally:

MAXIMAL SSG
<i>Input:</i> a node weighted digraph $G = (V, A, w)$ and a bound B .
<i>Output:</i> $S \subseteq V$ satisfying (1), (2) and (3).
<i>Objective:</i> minimize $w(S)$.

We also strengthen the digraph constraints by a new kind of constraints called *weak digraph constraints* and defined by:

$$N_G^-(x) \subseteq S \wedge N_G^-(x) \neq \emptyset \Rightarrow x \in S \quad (4)$$

By replacing (1) by (4) in the definition of SSG and MAXIMAL SSG, we obtain two additional optimization problems.

SUBSET SUM WITH WEAK DIGRAPH CONSTRAINTS

This problem is called SUBSET SUM WITH WEAK DIGRAPH CONSTRAINTS (SSGW in short).

SSGW
<i>Input:</i> a node weighted digraph $G = (V, A, w)$ and a bound B .
<i>Output:</i> $S \subseteq V$ satisfying (4) and (2).
<i>Objective:</i> maximize $w(S)$.

MAXIMAL SUBSET SUM WITH WEAK DIGRAPH CONSTRAINTS

As previously, we can define a maximal subset satisfying the weak digraph constraint:

$$\text{There is no } S' \supset S \text{ such that } S' \text{ satisfies (4) and (2).} \quad (5)$$

This condition is denoted by the *weak maximality constraint* and the last problem is called MAXIMAL SUBSET SUM WITH WEAK DIGRAPH CONSTRAINTS (MAXIMAL SSGW in short)

MAXIMAL SSGW
<i>Input:</i> a node weighted digraph $G = (V, A, w)$ and a bound B .
<i>Output:</i> $S \subseteq V$ satisfying (4), (2) and (5).
<i>Objective:</i> minimize $w(S)$.

The feasibility of a solution $S \subseteq V$ for MAXIMAL SSGW can be decided in $O(n^2)$. Indeed, (4) and (2) are checked in $O(n)$ and the maximality constraint (5) can be checked as follows: for every $v \in V \setminus S$, add v and (inductively) the vertices that allow to satisfy (4) in S (by necessary condition) and check condition (2) with the increased set because the weights are non-negative.

3 Related works

The SUBSET SUM PROBLEM is one of the simplest and fundamental **NP**-hard problems. It appears in many real world applications. Given n integers a_i for $i = 1, \dots, n$ and a target B , the goal is to find a subset $S \subseteq \{1, \dots, n\}$ such that $\sum_{i \in S} a_i = B$. A survey of existing results on the SUBSET SUM PROBLEM can be found in Chapter 4 of [23]. There are several generalizations of the SUBSET SUM PROBLEM studied in the literature, see for instance [27, 25, 12, 13, 5]. In [27], the variation, called EQUAL SUBSET SUM FROM TWO SETS is shown to be **NP**-complete, where given a set of n integers a_i for $i = 1, \dots, n$, the problem is to decide whether there exist two disjoint nonempty subsets of indices $S_1, S_2 \subseteq \{1, \dots, n\}$ such that $\sum_{i \in S_1} a_i = \sum_{j \in S_2} a_j$. In [25], two generalizations to intervals are proposed and they are motivated by single-item multi-unit auctions; here, we are given a set of n intervals $[a_i, b_i]$, a target B , and the goal is to choose a set of integers (at most one from each interval for the first problem and for the second problem, the additional restriction that at least k_1 and at most k_2 integers must be selected), whose sum approximates B as best as possible. Several results are proposed, including a **FPTAS**. In [13], the problem of deciding whether all integer values between two given bounds B^- and B^+ are attainable is proved to be Π_p^2 -complete.

Many variations of the SUBSET SUM PROBLEM have also been studied in [11, 12] and especially a version on undirected graphs called ESS WITH EXCLUSIONS. Given a connected undirected exclusion graph $G = (V, E)$ where the nodes are weighted by $w(v) \geq 0$, the problem consists in deciding if there are two disjoint independent sets $X, Y \subseteq V$ of G such that $w(X) = \sum_{x \in X} w(x) = \sum_{y \in Y} w(y)$. ESS WITH EXCLUSIONS is obviously **NP**-complete and a pseudo-polynomial time algorithm is presented in [11, 12].

The PARTIALLY-ORDERED KNAPSACK problem (also known as the PRECEDENCE - CONSTRAINED KNAPSACK PROBLEM) is a natural generalization of SSG (exactly as KNAPSACK generalizes SUBSET SUM). Here, we are given a set V of items, a DAG $G = (V, A)$ (or equivalently a poset \prec_P on V) and a bound B . Each item $v \in V$ has a size $p(v) \geq 0$ and an associated weight $w(v) \geq 0$. The objective is to find $S \subseteq V$, whose weight $w(S) = \sum_{v \in S} w(v)$ is maximized under the digraph constraints and also $p(S) := \sum_{v \in S} p(v)$ must be at most B . When $p(v) = w(v)$, we clearly obtain SSG. PARTIALLY-ORDERED KNAPSACK is strongly **NP**-hard, even when $p(v) = w(v)$, $\forall v \in V$, and G is a bipartite DAG [22]. In 2006, it was demonstrated in [19] that PARTIALLY-ORDERED KNAPSACK is hard to approximate within a factor $2^{(\log n)^\varepsilon}$, for some $\varepsilon > 0$, unless $3SAT \in \mathbf{DTIME}(2^{n^{\frac{3}{4}+\delta}})$. A survey of some applications and results can also be found in the book (pages 402-408 of [23]).

In [24], some **FPTAS** are proposed for PARTIALLY-ORDERED KNAPSACK in the case of 2-dimensional partial ordering (a generalization of series-parallel digraphs) and in the DAG whose

bipartite complement are chordal bipartite. Also, a polynomial-time algorithm for PARTIALLY-ORDERED KNAPSACK on Red-Blue bipartite DAG is given when its bipartite complement is chordal bipartite.

In the case of rooted trees, a **FPTAS** is also proposed for PRECEDENCE-CONSTRAINED KNAPSACK PROBLEM in [22]. The special case of in-rooted trees is also known in the literature as the *Tree Knapsack Problem* [4,10].

In [7] an approach based on clique inequalities is presented for determining facets of the polyhedron of the PRECEDENCE-CONSTRAINED KNAPSACK PROBLEM.

Two related problems, known as the NEIGHBOUR KNAPSACK PROBLEM, are studied in [8] in which dependencies between items are given by an undirected (or a directed) graph $G = (V, E)$. In the first version, an item can be selected only if at least one of its neighbors is also selected. In the second version, an item can be selected only when all its neighbors are also selected. The authors of [8] propose upper and lower bounds on the approximation ratios for these two problems on undirected and directed graphs.

Concerning the *Maximal version* of SUBSET SUM, this problem is called the LAZY BUREAUCRAT PROBLEM with common arrivals and deadlines in the literature [15,18] where the problem has been proved **NP**-hard and approximable with a **FPTAS**. This latter problem has also several generalizations known as the LAZY BUREAUCRAT SCHEDULING PROBLEM [2,14,15] and the LAZY MATROID PROBLEM [17].

4 Regular Digraphs

Theorem 1. *SSG is strongly NP-hard for connected digraphs in which each node has either out-degree 2 and in-degree 1 or the reverse.*

Proof. We prove the strong **NP**-hardness using a reduction from CLIQUE:

CLIQUE
<i>Input:</i> a connected simple graph $G = (V, E)$.
<i>Output:</i> $V' \subseteq V$ such that every two vertices in V' are joined by an edge in E .
<i>Objective:</i> maximize $ V' $.

CLIQUE is known to be **NP**-hard, even in regular connected graphs of degree $\Delta \geq 3$ (Problem [GT19], page 194 in [16]).

Let $I = (G, k)$ be an instance of the decision version of CLIQUE where $G = (V, E)$ is a regular connected graph of degree Δ and $V = \{1, \dots, n\}$. We construct an instance $I' = (G' = (V', A'), w, B)$ of SSG as follows:

$G' = (V', A')$ is a connected digraph defined by $V' = V_G \cup V_E$ where $V_G = \{v_{i,j} : i \in V, j \in N_G(i)\}$ and $V_E = \{v_e^i : e \in E, i = 1, \dots, 6\} = \cup_{e \in E} H(e)$ where $H(e) = \{v_e^i, i = 1, \dots, 6\}$ is a gadget. We start from G , and we replace each node $i \in V$ by a circuit C_i of Δ vertices $v_{i,j}^j$ for $j \in N_G(i)$. Then two circuits C_i and C_j are connected via a gadget $H(e)$ if edge $e = [i, j] \in E$, in such a way that each node of the circuit has in-degree 2 and out-degree 1. Formally, if $e = [x, y] \in E$, then the gadget $H(e)$ has 6 nodes $\{v_e^i : i = 1, \dots, 6\}$ where $v_e^1 = v_e^x$ and $v_e^6 = v_e^y$. An illustration is given in Figure 1.

If $e = [i, j] \in E$, then we add the two arcs $(v_e^i, v_{i,j})$ and $(v_{j,i}, v_e^j)$ in G' . Finally, each node of each circuit C_i has weight 1, i.e., $w(v_{i,j}) = 1$ while $w(v_e^i) = \Delta n$ for $i = 1, \dots, 6$. The bound $B = 3\Delta nk(k-1) + \Delta k$.

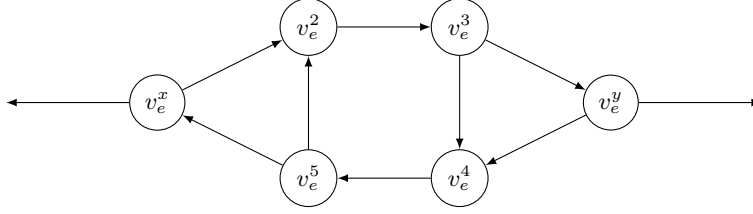


Fig. 1. The gadget $H(e)$ for $e = [x, y] \in E$.

An illustration of the construction is given in Figure 3 for the graph described in Figure 2. Clearly, this construction is done in polynomial time and G is a 3-regular connected digraph. Moreover, for each $v \in V'$ either $d_{G'}^+(v) = 2$ and $d_{G'}^-(v) = 1$, or $d_{G'}^-(v) = 2$ and $d_{G'}^+(v) = 1$.

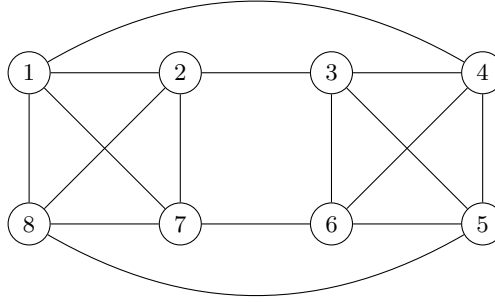


Fig. 2. Example of an instance G of CLIQUE with $\Delta = 4$.

We claim that there is a clique $S \subseteq \{1, \dots, n\}$ of size k if and only if there is $S' \subseteq V'$ satisfying the digraph constraints (1) in G' with $w(S') = B = 3\Delta nk(k-1) + \Delta k$.

Assume there exists a clique S of size $|S| = k$ in G . Then the subgraph induced by S contains $\frac{k(k-1)}{2}$ edges. The set $S' = (\cup_{i \in S} C_i) \cup \{v_e^\ell \in V_E : e = [i, j] \in E(S), \ell = 1, \dots, 6\}$ satisfies the digraph constraints (1) in G' and $w(S') = 6\Delta n \frac{k(k-1)}{2} + \Delta|S| = 3\Delta nk(k-1) + \Delta k = B$.

Conversely, assume there exists $S' \subseteq V'$ satisfying the digraph constraints (1) with $w(S') = B = 3\Delta nk(k-1) + \Delta k$ for some integer $k \geq 2$. Then S' contains $\frac{k(k-1)}{2}$ gadgets $H(e)$ (because each $H(e)$ is strongly connected) with total weight $6\Delta n$ and Δk vertices from V with weight 1 since the weights of vertices of $V' \setminus V_E$ cannot compensate the weights of one vertex of V_E . Due to the digraph constraints, for every gadget $H(e) \subseteq S' \cap V_E$, the two circuits C_i, C_j such that $e = [i, j]$ must entirely belong to S' . We construct the set $S = \{i : C_i \subseteq S'\}$. Then S contains exactly k vertices. In addition, the subgraph G_S of G induced by S contains $\frac{k(k-1)}{2}$ edges representing the $\frac{k(k-1)}{2}$ gadgets $H(e)$ in $S' \cap V_E$. We conclude that G_S is a complete graph, so S is a clique of size k in G . ■

Corollary 1. CARDINALITY SSG is strongly NP-complete in general digraphs even if the weight of each node is either a or b with $1 \leq a < b$ integers.

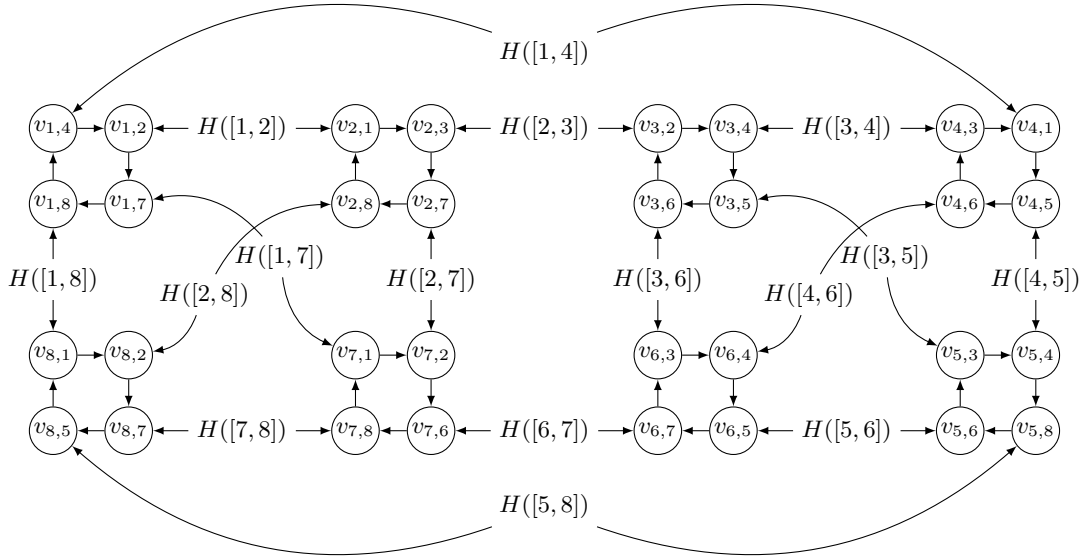


Fig. 3. Example of digraph G' constructed in the reduction from G .

Proof. Using the reduction proposed in the proof of Theorem 1, we can see that the weight of each node of G' is either 1 or Δn and the size of S is exactly $p = \Delta k + 6 \frac{k(k-1)}{2}$ where k is the size of the clique. ■

We now prove that Theorem 1 is the best possible complexity result according to degree parameters, that is either SSG is **NP**-hard in connected digraphs of maximum degree 2 or maximum out-degree 1 or in-degree 1, but admits a pseudo polynomial algorithm for these digraphs or SSG is polynomial for connected digraphs with in-degree and out-degree 2 for each node.

Lemma 1. *SSG is polynomial-time solvable in connected digraphs for which the in-degree and the out-degree of each node is 2.*

Proof. Let $I = (G, w, B)$ be an instance of SSG such that $G = (V, A)$ is a connected digraph where the in-degree and the out-degree of each node is exactly 2.

Using Euler-Hierholzer Theorem [6], we know that G is Eulerian, i.e. there is a circuit visiting each arc of A exactly once. Hence, by the digraph constraints (1), a feasible solution is either the empty set or the whole set of vertices V . Since w is non-negative, it follows that V is an optimal solution if and only if $w(V) \leq B$. ■

For the other cases, the digraph is a chain or an oriented tree (see Propositions 1 and 2, page 13, and Remark 4, page 14) and then belongs to the class of *Directed Acyclic Graphs* (DAG in short).

5 Directed Acyclic Graphs

Let us start by some definitions and notions useful in the rest of this section. Given a connected DAG $G = (V, A)$, we say that $v' \in V$ is a *descendant* of v iff $v = v'$ or there is a directed path

from v to v' in G . Let $desc_G(v)$ denote the set of descendants of v in G . An *ascendant* of v is a node $u \neq v$ such that $v \in desc_G(u)$. The set of ascendants of v in G is denoted by $asc_G(v)$. Obviously, $desc_G(v) \cap asc_G(v) = \emptyset$ because G is a DAG. By extension, given $S \subseteq V$, $desc_G(S) = \bigcup_{v \in S} desc_G(v)$ and $asc_G(S) = \bigcup_{v \in S} asc_G(v)$. Clearly, a solution S satisfies the digraph constraints (1) iff $S = desc_G(S)$.

The *kernel* of a subset $S \subseteq V$, denoted by $\kappa(S)$, is a subset of minimal size such that:

1. $\kappa(S) \subseteq S$
2. $desc_G(\kappa(S)) = desc_G(S)$.

Because G is a DAG, the notion of kernel is well defined and unique (actually, $\kappa(S)$ is the set of sources of the subgraph $G[S]$ induced by S). Note that $\kappa(S)$ is an independent set of G . The notion of $\kappa(S)$ is important because it constitutes in some sense the core of the digraph constraints. The following properties can be easily checked:

Property 1. Let S be a subset of a DAG $G = (V, A)$ satisfying the digraph constraints (1). Let $v \in V$:

- $v \in S$ iff $desc_G(v) \subseteq S$
- $v \notin S$ implies $asc_G(v) \cap S = \emptyset$

Now, we show that SSG and MAXIMAL SSG in general digraphs can be restricted to DAG.

Lemma 2. *The resolution (or approximation) of SSG (MAXIMAL SSG resp.) in general digraphs and connected DAG are equivalent.*

Proof. Take a digraph G , instance of SSG (MAXIMAL SSG resp.) and replace each strongly connected component by a representative vertex whose weight is the sum of the weights of the nodes that it represents. It is known that the resulting graph G' , called the *condensation* of G , is a DAG. It is not difficult to see that there is a bijection between the feasible solutions in G and the feasible solutions in G' . Moreover, the weight of the solution is preserved. ■

Corollary 2. *CARDINALITY SSG is strongly NP-complete in DAG even if the weight of each node is either a or b with $1 \leq a < b$ integers.*

Proof. Using both reductions proposed in the proofs of Theorem 1 and Lemma 2, we can see that the weight of each node of G' is either 1 or $6n$ (Δ and $6\Delta n$ simplified to 1 and $6n$) and the size of S is exactly $p = k + \frac{k(k-1)}{2}$ where k is the size of the clique. ■

Remark 1.

1. CARDINALITY SSG is polynomial if all the nodes have the same weight a .
2. CARDINALITY SSG is strongly NP-complete if the weight of a node is either 0 or $a > 0$.
3. SSG is polynomial-time solvable when there is a unique weight a or two weights 0 and $a > 0$.

Proof.

1. In the case of a unique weight $a \geq 0$, if $|V| * a < B$ or $B \neq p * a$, then the answer is no. Otherwise, start from $S = \emptyset$ and iteratively add to S a sink of $G[V \setminus S]$.
2. If the weight of each node is either 0 or $a > 0$, we use the same reduction as in the proof of Corollary 2 and we replace the weights Δ and $6\Delta n$ by 1 and 0 respectively.
3. In the case of SSG, start from $S = \emptyset$ and iteratively add to S a sink of $G[V \setminus S]$. ■

Lemma 3. *Let $I = (G, w, B)$ be an instance of MAXIMAL SSG (MAXIMAL SSGW, resp.) such that G is a DAG. $S \subseteq V$ is a feasible solution to MAXIMAL SSG if and only if S satisfies (1), (2) and (6).*

$$\forall x \in V \setminus S, \quad S \cup \{x\} \text{ violates (1) or (2)} \quad (6)$$

Similarly, $S \subseteq V$ is a feasible solution to MAXIMAL SSGW if and only if S satisfies (4), (2) and (7).

$$\forall x \in V \setminus S, \quad S \cup \{x\} \text{ violates (4) or (2)} \quad (7)$$

Proof. The proof is only detailed for MAXIMAL SSG. Similar arguments can be used for MAXIMAL SSGW. Let $I = (G, w, B)$ be an instance of MAXIMAL SSG such that $G = (V, A)$ is a DAG.

By definition, a feasible solution to MAXIMAL SSG satisfies (1), (2) and (3). Since (3) is stronger than (6), one direction of the equivalence (i.e. \Rightarrow) holds trivially.

For the other direction, take any $S \subseteq V$ satisfying (1), (2) and (6). By contradiction, suppose there exists $S' \supset S$ such that S' satisfies (1) and (2). Since G is a DAG, the sub-graph G' induced by $S' \setminus S$ is a non-empty DAG. So G' contains a sink $v \in S' \setminus S$. By definition, $S \cup \{v\}$ satisfies the digraph constraints (1). Moreover, $S \cup \{v\}$ satisfies the budget constraint (2) because $w(S \cup \{v\}) \leq w(S') \leq B$. We get a contradiction with (6). ■

Remark 2. Lemma 3 is equivalent to check that for a sink v of minimum weight (among the sinks of $G[V \setminus S]$), $w(B) + w(v) > B$.

Remark 3. Using (6), one can verify if a set $S \subseteq V$ is feasible for MAXIMAL SSG in $O(m)$.

The LAZY BUREAUCRAT PROBLEM with a common release date and a common deadline has been proved (weakly) **NP**-hard [15] and admitting a pseudo-polynomial algorithm [14]; recently, a **FPTAS** was proposed in [18]. Here, we prove that MAXIMAL SSG, a generalization of the lazy bureaucrat problem, is much harder.

Theorem 2. *MAXIMAL SSG is Strongly **NP**-hard in connected DAG with a unique sink and 3 distinct weights.*

Proof. We propose a Karp reduction from CARDINALITY SSG proved strongly **NP**-complete even with 2 distinct weights in Corollary 2.

Given an instance $I = (G, w, p, B)$ of CARDINALITY SSG where $G = (V, A)$ is a DAG, $V = \{v_1, \dots, v_n\}$, $1 \leq w(v) \leq B - p$ and 2 distinct weights, we build an instance $I' = (G', w', B', q)$ of the decision version of MAXIMAL SSG (I is a yes-instance if there exists a feasible solution of weight at most q) where $G' = (V', A')$ is a connected DAG with a unique sink as follows:

- $G' = (V', A')$ has $n + p + 1$ vertices $V' = V \cup D$ where $D = \{v_{n+i} : i = 1, \dots, p + 1\}$ and contains G as a subgraph. If S denotes the set of sinks of G , then $A' = A \cup \{(u, v_{n+1}) : u \in S\} \cup \{(v_{i+1}, v_i) : i = n + 1, \dots, n + p\}$.
- $w'(v) = p^2 B + pB + w(v)$ for $v \in V$ while $w'(v_{n+i}) = p^2 B$ for $i = 1, \dots, p + 1$. Finally, $B' = p^3 B + 3p^2 B + B - 1$.
- $q = p^3 B + 2p^2 B + B$.

Figure 4 gives an illustration of this construction. G' is a connected DAG with unique sink and $w'(v)$ can take at most 3 distinct values. These values are positive integers. Obviously, this reduction can be done in polynomial time, and all values $w'(v)$ and B' are upper bounded by a polynomial because $w(v)$ and B are as such.

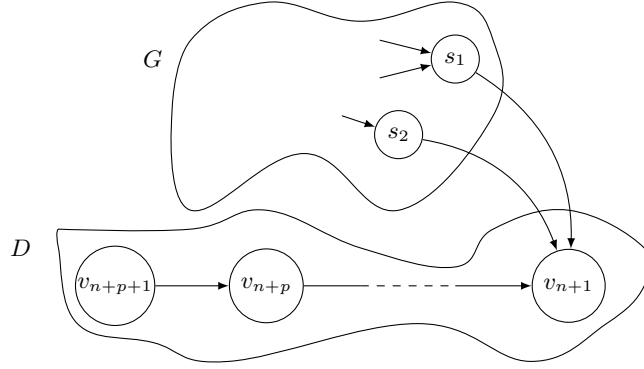


Fig. 4. The graph G' .

We claim that I is a yes-instance of **CARDINALITY SSG**, that is $\exists J \subset V$ with $|J| = p$, J satisfies (1) and $w(J) = B$ iff $\exists J' \subset V'$ such that $w'(J') \leq q = p^3B + 2p^2B + B$ and $w'(J'') > B'$ for every J'' of V' satisfying the digraph constraints and containing J' .

Clearly, if I is a yes-instance of **CARDINALITY SSG**, then by definition there is $J \subseteq V$ with $|J| = p$ such that J satisfies (1) and $\sum_{v \in J} w(v) = B$. Hence for **MAXIMAL SSG**, the subset $J' = J \cup \{v_{n+1}\}$ has weight $\sum_{u \in J'} w'(u) = p^2B + p(p^2B + pB) + \sum_{v \in J} w(v) = p^3B + 2p^2B + B = q$. This set also satisfies the maximality constraint because using Property (6) of Lemma 3, we know that the addition to J' of any sink u of the subgraph of G' induced by $V' \setminus J'$ gives $w'(J' \cup \{u\}) \geq q + p^2B = p^3B + 3p^2B + B > B'$ since $w'(v) \geq p^2B$ for all $v \in V'$.

Conversely, let $J' \subset V'$ be a feasible solution of **MAXIMAL SSG** in G' with weight $\sum_{v \in J'} w'(v) \leq q = p^3B + 2p^2B + B$. First, let us show that (i) $v_{n+1} \in J'$ and $D \not\subseteq J'$, (ii) this sum $w(J') = q = p^3B + 2p^2B + B$ and (iii) $|J'| = p + 1$ and $J' \cap D = \{v_{n+1}\}$.

- For (i). By the maximality constraint and because v_{n+1} is the unique sink of G' , we must have $v_{n+1} \in J'$. Now by contradiction, suppose $D \subseteq J'$. Then, let us prove that $J' \cap V \neq \emptyset$ because otherwise $J' = D$. Since $G = (V, A)$ is a DAG, there exists a sink $u \notin J'$ of G . The maximality constraint on G' is not satisfied because $B' - w'(J') = (p^3B + 3p^2B + B - 1) - (p + 1)p^2B = 2p^2B + B - 1 > w'(u)$. Hence, u should be added to J' . In conclusion $D \cup \{u\} \subseteq J'$ and we deduce $w'(J') \geq w'(D \cup \{u\}) \geq (p + 1)p^2B + p^2B + pB + 1 > p^3B + 2p^2B + B = q$ which is a contradiction with the initial hypothesis.
- For (ii). Using (i), we know that there exists $v_{n+\ell} \notin J'$ and $v_{n+\ell-1} \in J'$ for some $\ell \in \{2, \dots, p + 1\}$. Hence, the maximality constraint imposes that $v_{n+\ell}$ is a sink in the subgraph of G' induced by $V' \setminus J'$, so $w'(J') \geq B' - w'(v_{n+\ell}) + 1 = p^3B + 2p^2B + B = q$.
- For (iii). First, observe that $|J' \setminus \{v_{n+1}\}| = p$ because on the one hand if $|J'| \leq p$, then $w'(J') \leq p^2B + (p - 1)(p^2B + pB + B) < p^3B + p^2B - B < q$ and on the other hand, if $|J'| \geq p + 2$, then $w'(J') > p \times p^2B + 2(p^2B + pB + 1) = p^3B + 2p^2B + 2pB + 2 > p^3B + 2p^2B + B = q$, because for both cases by item (i) we know $|J' \cap D| \leq p$ and the weights of nodes in D are the smallest of G' . These two cases lead to a contradiction with item (ii). Finally, let us prove that $J' \cap D = \{v_{n+1}\}$. Otherwise, $|J' \cap V| \leq p - 1$. Since the worst case

appears when $|J' \cap V| = p - 1$, then $w'(J') < ((p - 1)(p^2B + pB) + (p - 1)B) + 2p^2B = p^3B + 2p^2B - B < p^3B + 2p^2B + B = q$, which is another contradiction with item (ii).

Using (i), (ii) and (iii), and by setting $J = J' \setminus \{v_{n+1}\}$ we deduce $J \subseteq V$, J satisfies the digraph constraints and $|J| = p$ with $w(J) = w'(J') - p(p^2B + pB) - p^2B = q - p^3B - 2p^2B = B$. Hence, I is a yes-instance of CARDINALITY SSG. ■

We now prove that the weak digraph constraints versions of the two problems are as hard to approximate as two variants of the INDEPENDENT SET problem (IS in short):

IS
<i>Input:</i> a connected simple graph $G = (V, E)$.
<i>Output:</i> $V' \subset V$ such that no two vertices in V' are joined by an edge in E .
<i>Objective:</i> maximize $ V' $.

The *independence number* of G , denoted by $\alpha(G)$, is the maximum size of an independent set in G . IS is known to be **APX**-complete in cubic graphs [1], **NP**-hard in planar graphs with maximum degree $\Delta(G) \leq 3$ [16] and no polynomial algorithm can approximately solve it within the ratio $n^{\varepsilon-1}$ for any $\varepsilon \in (0; 1)$, unless **P=ZPP** [21].

The second problem is the MINIMUM INDEPENDENT DOMINATING SET problem (ISDS in short) also known as the MINIMUM MAXIMAL INDEPENDENT SET problem (Problem [GT2], see comment page 190 in [16]).

ISDS
<i>Input:</i> a connected simple graph $G = (V, E)$.
<i>Output:</i> $V' \subset V$ such that no two vertices in V' are joined by an edge in E and for all $u \in V \setminus V'$, there is a $v \in V'$ for which $(u, v) \in E$.
<i>Objective:</i> minimize $ V' $.

The *independent domination number* of G , denoted by $i(G)$ is the minimum size of an independent dominating set in G . Obviously, $i(G) \leq \alpha(G)$. ISDS is known to be **NP**-hard [16], even for planar cubic graphs [26] and it is **APX**-complete for graphs of maximum degree 3 [9]. Furthermore, it is also very hard from an approximation point of view, since no polynomial algorithm can approximately solve it within the ratio $n^{1-\varepsilon}$ for any $\varepsilon \in (0; 1)$, unless **P=NP** [20].

Theorem 3. *There is a polynomial reduction preserving approximation from*

1. IS in general graphs to SSGW in DAG with maximum in-degree 2.
2. ISDS in general graphs to MAXIMAL SSGW in DAG with maximum in-degree 2.

Proof. Let G be a connected graph where $G = (V, E)$ with $V = \{v_1, \dots, v_n\}$. We construct a corresponding instance $I = (G' = (V', A'), w, B)$ of SSGW and MAXIMAL SSGW as follows: Let $G' = (V', A')$ be a digraph defined by $V' = V \cup V_E$ where $V_E = \{v_e : e \in E\}$ and $A' = \{(v_i, v_e) : e = [v_i, v_j] \in E\}$. Set $w(v_i) = 1$ for $i = 1, \dots, n$ and $w(v_e) = n + 1$ for all $v_e \in V_E$ and $B = n$. Clearly, G' is a connected DAG with maximum in-degree 2. An example of such reduction is given in Figure 5.

We claim that S is a maximal independent set of G if and only if the same set of vertices in digraph G' is a feasible solution to SSGW (MAXIMAL SSGW resp.).

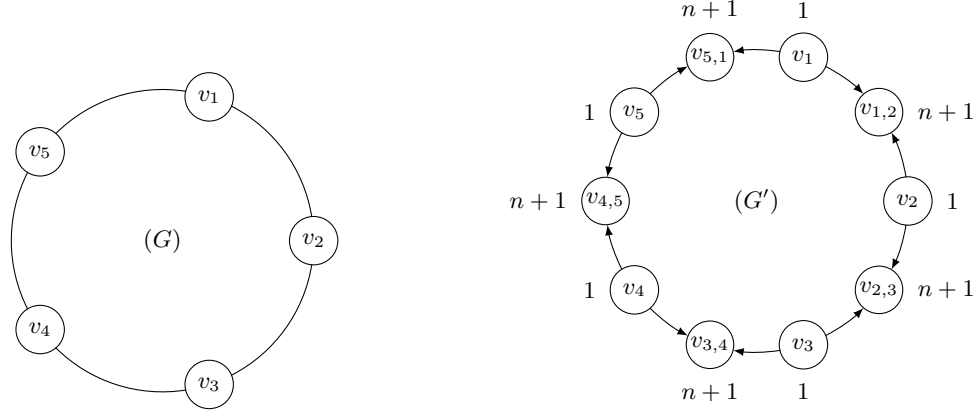


Fig. 5. Example of reduction.

S is a maximal independent set of G iff this subset satisfies the weak digraph constraints in G' (otherwise, v_e for some $e = [x, y]$ with $x, y \in S$ should be added to S) and using Lemma 3, we can not add a new vertex because on the one hand, $w(S) = |S| \leq n$ and on the other hand, either $S + v$ does not satisfies the weak digraph constraints for $v \notin S$ by maximality of S or $w(S + v_e) > n$ for some $e \in E$.

For item 1, the result follows from the definition of IS while for item 2 it is the the definition of ISDS. ■

Corollary 3. *In DAG with maximum in-degree is 2, we have:*

1. SSGW and MAXIMAL SSGW are **APX**-hard even if the maximum out-degree is 3.
2. For every $\varepsilon \in (0, 1/2)$, SSGW (MAXIMAL SSGW resp.) is not $n^{\varepsilon - \frac{1}{2}}$, unless $\mathbf{P} = \mathbf{ZPP}$ ($n^{\frac{1}{2} - \varepsilon}$ unless $\mathbf{P} = \mathbf{NP}$ resp.).

Proof. For item 1, it is a consequence of Theorem 3 together with the results of [1,9].

For item 2, we use the negative results given in [21,20] and the reduction of Theorem 3 with $|V(G')| = |E(G)| + |V(G)| \leq |V(G)|^2$. ■

Lemma 4. *SSG and MAXIMAL SSG are polynomial-time solvable in tournaments.*

Proof. Let G be a tournament digraph. We may assume that G is acyclic by Lemma 2. Thus G contains a unique Hamiltonian path [3]. We denote it by $\mathcal{H} = \{(v_i, v_{i+1}) : i = 1, \dots, n-1\}$ where n is the number of vertices of G . Due to the digraph constraints, a feasible solution is, either the empty set, or a subset $\{(v_i, v_{i+1}) : i = k, \dots, n-1\}$ for some $k \in \{1, \dots, n-1\}$ because G is an acyclic tournament and then $(v_i, v_j) \notin A$ for $j < i$. Once \mathcal{H} is found (this can be done in polynomial time), we can make a binary search on \mathcal{H} to find a solution if it exists in $O(n \log n)$. ■

6 Oriented Trees

Proposition 1. *The four following problems are **NP**-hard in out-rooted trees and in in-rooted trees:*

1. SSG and SSGW.
2. MAXIMAL SSG and MAXIMAL SSGW.

Proof. We only prove the case of out-rooted trees (for in-rooted trees, we reverse the orientation of each arc).

For item 1, we show the **NP**-hardness using a reduction from SUBSET SUM (Problem [SP13], page 223 in [16]) known to be (weakly) **NP**-complete. This problem is described as follows:

SUBSET SUM (SS)
<i>Input:</i> a finite set X , a size $s(x) \in \mathbb{Z}^+$ for each $x \in X$ and a positive integer B .
<i>Question:</i> is there a subset $S \subseteq X$ such that $s(S) = \sum_{x \in S} s(x) = B$?

Let $I = (X, s, B)$ be an instance of SS. We polynomially construct a corresponding instance $I' = (T, w, B, k)$ of the decision version SSG and SSGW where k is an integer. Let $T = (V, A)$ be a digraph defined by $V = X \cup \{r\}$ and $A = \{(v, r) : v \in X\}$. Clearly, T is an out-rooted tree. The weight function is $w(v) = s(v)$ for all $v \in X$ and $w(r) = 0$. Finally, we set $k = B$.

It is easy to show for SSG (SSGW resp.) that S is a solution of SUBSET SUM if and only if $S \cup \{r\}$ satisfies the digraph constraints (1) (the weak digraph constraints (4) resp.) and $w(S \cup \{r\}) \geq k$.

For item 2, we prove the **NP**-hardness using a reduction from the LAZY BUREAUCRAT PROBLEM with common deadlines and release dates. The decision version of this problem has been shown **NP**-complete in [15] and it can be described by:

DECISION LAZY BUREAUCRAT
<i>Input:</i> a finite set X , a size $s(x) \in \mathbb{Z}^+$ for each $x \in X$, positive integers B and $k \leq B$.
<i>Question:</i> is there a subset $S \subseteq X$ such that $s(S) = \sum_{x \in S} s(x) \leq k$ and $\forall x \notin S, s(S) + s(x) > B$?

Let $I = (X, s, B, k)$ be an instance of DECISION LAZY BUREAUCRAT. We construct an instance $I' = (T, w, B, k)$ in the same way as for item 1. Clearly, there is a subset $S \subseteq X$ with $s(S) = \sum_{x \in S} s(x) \leq k$ and $\forall x \notin S, s(S) + s(x) > B$ if and only if $S \cup \{r\}$ satisfies the (weak resp.) digraph constraints with $w(S \cup \{r\}) \leq k$ and $\nexists S' \supset S$ that satisfies the (weak resp.) digraph constraints with $w(S' \cup \{r\}) \leq B$. ■

Remark 4. The reduction of Proposition 1 can also be modified in order to get $w(v) > 0$ for every vertex v . Moreover, we can slightly modify the construction in order to obtain a binary tree or a chain.

We now present some dynamic programs for solving the different problems defined in Section 2.2 in the class of trees.

Beforehand, we introduce some notations on trees. Let $T = (V, A)$ be a directed tree. Let us fix any vertex $r(T) \in V$ as the root of the underlying tree T .

For a node $v \in V$, we denote by $fa(v)$ its father and by $ch(v)$ its set of children. The root has no father and its neighbors are its children. For a node v that is not $r(T)$, its father $fa(v)$ is the first node of the unique path from v to $r(T)$ (with $v \neq fa(v)$). Note that this path is not necessarily directed, i.e. an arc can be traversed from head to tail or the other way. The children set of v , denoted by $ch(v)$, is defined as $N_T(v) \setminus \{fa(v)\}$.

We partition the set $ch(v)$ of children of v into two sets $ch^+(v)$ and $ch^-(v)$. The set $ch^+(v)$ contains the children of v in T that leave v , i.e. $ch^+(v) = \{u \in ch(v) : (v, u) \in A\}$. The set $ch^-(v)$ contains the children of v in T that arrive in v , i.e. $ch^-(v) = \{u \in ch(v) : (u, v) \in A\}$.

For a child i of the root $r(T)$, let V_i denote the nodes accessible from i without passing through $r(T)$. That is, V_i consists of vertex i , the children of i , the children of these children, etc. Let T_i denote the tree induced by V_i in which i is the root.

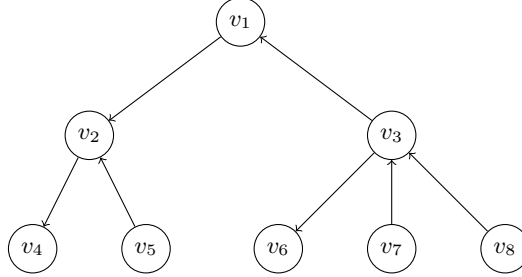


Fig. 6. Example of a directed tree.

An example of a directed tree $T = (V, A)$ is given in figure 6. Let us fix arbitrarily the vertex v_1 as the root $r(T)$ of the tree T . Then $fa(v_1) = \emptyset$, $ch(v_1) = \{v_2, v_3\}$, $ch^+(v_1) = \{v_2\}$ and $ch^-(v_1) = \{v_3\}$. Regarding vertex v_3 , $fa(v_3) = \{v_1\}$, $ch(v_3) = \{v_6, v_7, v_8\}$, $ch^+(v_3) = \{v_6\}$ and $ch^-(v_3) = \{v_7, v_8\}$. Also, $ch(v_i) = \emptyset$ for $i \in \{4, 5, 6, 7, 8\}$. Moreover, $V_{v_2} = \{v_2, v_4, v_5\}$ and T_{v_2} is the sub-tree induced by V_{v_2} . In the same way, T_{v_3} is the sub-tree induced by $V_{v_3} = \{v_3, v_6, v_7, v_8\}$. Observe that $V = \{r(T)\} \cup_{i \in ch(r(T))} V_i$.

Proposition 2. SSG can be solved using dynamic programming in oriented trees.

Proof. Let $I = (T, w, B)$ be an instance of SSG where $T = (V, A)$ is a directed tree with a root $r(T) \in V$. Given an integer $b \in \{0, \dots, B\}$, let $R^+(T, b)$ ($R^-(T, b)$ resp.) be the boolean defined by $R^+(T, b) = True$ ($R^-(T, b) = True$ resp.) if and only if there exists $S \subseteq V$ satisfying (1) in the tree T with $r(T) \in S$ ($r(T) \notin S$ resp.) and $w(S) = b$. Let $R(T, b) = R^+(T, b) \vee R^-(T, b)$. Then $R(T, b)$ is *True* if and only if there exists $S \subseteq V$ satisfying (1) and $w(S) = b \leq B$, so S also satisfies (2). Hence it is feasible for SSG with the weight $w(S) = b$. We define $R^+(T, b)$ and $R^-(T, b)$ recursively as follows¹:

$$R^+(T, b) = \begin{cases} b == w(r(T)), & \text{when } V = \{r(T)\}, \\ \bigwedge_{k \in ch^+(r(T))} R^+(T_k, a_k) \bigwedge_{\ell \in ch^-(r(T))} R(T_\ell, b_\ell), & \text{when } |V| > 1 \text{ and where} \\ & a_k \geq 0, \forall k \in ch^+(r(T)), \\ & b_\ell \geq 0, \forall \ell \in ch^-(r(T)) \text{ and} \\ & \sum_{k \in ch^+(r(T))} a_k + \sum_{\ell \in ch^-(r(T))} b_\ell = b - w(r(T)) \end{cases}$$

¹ The formula "==" is the boolean test of equality.

$$R^-(T, b) = \begin{cases} b = 0, & \text{when } |V| = 1, \\ \bigwedge_{k \in ch^-(r(T))} R^-(T_k, c_k) \bigwedge_{\ell \in ch^+(r(T))} R(T_\ell, d_\ell), & \text{when } |V| > 1 \text{ and where} \\ & c_k \geq 0, \forall k \in ch^-(r(T)), \\ & d_\ell \geq 0, \forall \ell \in ch^+(r(T)) \text{ and} \\ & \sum_{k \in ch^+(r(T))} c_k + \sum_{\ell \in ch^-(r(T))} d_\ell = b \end{cases}$$

and $R(T, b) = R^+(T, b) \vee R^-(T, b)$.

Let us prove that R is well-defined or equivalently that R^+ and R^- are well-defined by induction on $|V|$. If $|V| = 1$ then $V = \{r(T)\}$. In this case, the unique feasible solution for R^- is the empty set, so $R^-(T, 0) = \text{True}$ and otherwise $R^-(T, b) = \text{False}$ for $b \neq 0$ and $b \leq B$. Regarding R^+ , the unique solution is reduced to the vertex $\{r(T)\}$, then $R^+(T, w(r(T))) = \text{True}$ and $R^+(T, b) = \text{False}$ for $b \neq w(r(T))$ and $b \leq B$.

Now, assume that $|V| \geq 2$. Then $r(T)$ has at least one child. Suppose that $R(T, b) = \text{True}$, i.e. $R^+(T, b) = \text{True}$ or $R^-(T, b) = \text{True}$ for some $b > 0$ (the case $b = 0$ corresponds to the empty solution).

- If $R^+(T, b) = \text{True}$ then there exists $S \subseteq V$ satisfying (1) in T with $r(T) \in S$ and $w(S) = b \leq B$. It follows that $k = r(T_k) \in S$ for all $k \in ch^+(r(T))$ because of (1). In addition, $S \cap V_k$ necessarily satisfies (1) in the sub-tree T_k . By setting $a_k = w(S \cap V_k)$, we conclude that $R^+(T_k, a_k) = \text{True}$ for all $k \in ch^+(r(T))$. Moreover, for all $\ell \in ch^-(r(T))$, $S \cap V_\ell$ satisfies (1) in T_ℓ . By setting $b_\ell = w(S \cap V_\ell)$, we get that $R(T_\ell, b_\ell) = \text{True}$ for all $\ell \in ch^-(r(T))$. Finally, $S = \{r(T)\} \cup_{k \in ch^+(r(T))} (S \cap V_k) \cup_{\ell \in ch^-(r(T))} (S \cap V_\ell)$, so $w(S) = w(r(T)) + \sum_{k \in ch^+(r(T))} a_k + \sum_{\ell \in ch^-(r(T))} b_\ell = b$ which is consistent with the definition of R^+ .
- If $R^-(T, b) = \text{True}$ then there is $S \subseteq V$ satisfying (1) in T with $r(T) \notin S$ and $w(S) = b \leq B$. Hence, for all $k \in ch^-(r(T))$, $k \notin S \cap V_k$ because of (1). Using the fact that $S \cap V_k$ satisfies (1) in the sub-tree T_k , it follows that $R^-(T_k, c_k) = \text{True}$ where $c_k = w(S \cap V_k)$ for all $k \in ch^-(r(T))$. Moreover, for all $\ell \in ch^+(r(T))$, $S \cap V_\ell$ satisfies (1) in the sub-tree T_ℓ , so $R(T_\ell, w(S \cap V_\ell)) = \text{True}$. The result follows by setting $d_\ell = w(S \cap V_\ell)$ for all $\ell \in ch^+(r(T))$.

Conversely, we denote by $S_i \subseteq V_i$ a solution satisfying $R(T_i, w(S_i)) = \text{True}$ for $i \in ch(r(T))$.

- If for all $k \in ch^+(r(T))$, $R^+(T_k, a_k) = \text{True}$ with $a_k \geq 0$, we know that for every $\ell \in ch^-(r(T))$, there exists $b_\ell \geq 0$ such that $R(T_\ell, b_\ell) = \text{True}$ (in the worst case, choose $b_\ell = 0$ because $R^-(T_\ell, 0)$ is always True); then we have $\{r(T)\} \cup_{k \in ch^+(r(T))} S_k \cup_{i \in I} S_i$ satisfies (1) in T . By setting $b = w(r(T)) + \sum_{k \in ch^+(r(T))} a_k + \sum_{\ell \in ch^-(r(T))} a_\ell$, we get that $R^+(T, b) = \text{True}$. Otherwise (if there exists $k \in ch^+(r(T))$ such that $R^+(T_k, a_k) = \text{False}$, $\forall a_k \geq 0$), we get that $R^+(T, b) = \text{False}$ for every $b \geq 0$ because of (1). Indeed, the root $r(T)$ can not belong to a feasible solution in T when it has a child $k \in ch^+(r(T))$ which is not in this solution.
- Moreover, if for all $k \in ch^-(r(T))$, $R^-(T_k, c_k) = \text{True}$ with $c_k \geq 0$, we know that for every $\ell \in ch^+(r(T))$, there exists $d_\ell \geq 0$ such that $R(T_\ell, d_\ell) = \text{True}$ (in the worst case, choose $d_\ell = 0$ because $R^-(T_\ell, 0)$ is always True); then we have $\{r(T)\} \cup_{k \in ch^-(r(T))} S_k \cup_{\ell \in ch^+(r(T))} S_\ell$

satisfies (1) in T . Let $b = \sum_{k \in ch^-(r(T))} c_k + \sum_{\ell \in ch^+(r(T))} d_\ell$. Then $R^-(T, b) = True$. Otherwise, if there exists $k \in ch^-(r(T))$ such that $R^-(T_k, a_k) = False, \forall a_k \geq 0$, then there is no solution S_k satisfying (1) in T_k with $w(S_k) = a_k$ and such that $k \notin S_k$. Hence, for every solution S of T , S must contain k , and by (1), it must contain $r(T)$, so $R^-(T, b) = False$ for every $b > 0$.

The value of R can easily be deduced from R^+ and R^- . The induction is proved.

R can be computed in $O(nB^2)$. Indeed, we can construct two tables, one for R^+ and the other one for R^- . In both tables, the columns are valued by the integers $0, 1, \dots, B$ and the lines contain sub-trees constructed as follows. We start by adding, in both tables, one line per leaf of T and for each graph T_ℓ induced by leaf ℓ , set $R^-(T_\ell, b) = True$ if and only if $b = 0$, and $R^+(T_\ell, b) = True$ if and only if $b = w(\ell)$. Then, add new lines containing new sub-trees with the following algorithm.

1. If there exist two sub-trees T_i and $T_{i'}$ induced by V_i and $V_{i'}$ respectively with $fa(i) = fa(i')$ whose lines have already been created in the tables then
 - (a) If there exists another child $i'' \neq i, i'$ such that $fa(i'') = fa(i) = fa(i')$ then choose i and i' such that both are either in $ch^+(fa(i))$ or in $ch^-(fa(i))$ and add a new line associated with the forest $T_{i,i'}$ induced by $V_i \cup V_{i'}$.
 - i. If $i, i' \in ch^+(fa(i))$ then
 - $R^+(T_{i,i'}, b) = True$ for $b \in \{0, \dots, B\}$ if and only if there exist $b_i, b_{i'} \in \{0, \dots, b\}$ such that $b = b_i + b_{i'}$ and $R^+(T_i, b_i) = R^+(T_{i'}, b_{i'}) = True$.
 - $R^-(T_{i,i'}, b) = True$ for $b \in \{0, \dots, B\}$ if and only if there exist $b_i, b_{i'} \in \{0, \dots, b\}$ such that $b = b_i + b_{i'}$ and $R(T_i, b_i) = R(T_{i'}, b_{i'}) = True$.
 - ii. If $i, i' \in ch^-(fa(i))$ then
 - $R^+(T_{i,i'}, b) = True$ for $b \in \{0, \dots, B\}$ if and only if there exist $b_i, b_{i'} \in \{0, \dots, b\}$ such that $b = b_i + b_{i'}$ and $R(T_i, b_i) = R(T_{i'}, b_{i'}) = True$.
 - $R^-(T_{i,i'}, b) = True$ for $b \in \{0, \dots, B\}$ if and only if there exist $b_i, b_{i'} \in \{0, \dots, b\}$ such that $b = b_i + b_{i'}$ and $R^-(T_i, b_i) = R^-(T_{i'}, b_{i'}) = True$.
 - (b) Else (there is no other child $i'' \neq i, i'$ with the same father $fa(i'') = fa(i) = fa(i')$), add a new line associated with the tree $T_{fa(i)}$ induced by $V_i \cup V_{i'} \cup \{fa(i)\}$.
 - i. If $i, i' \in ch^+(fa(i))$ then
 - $R^+(T_{fa(i)}, b) = True$ for $b \in \{0, \dots, B\}$ if and only if there exist $b_i, b_{i'} \in \{0, \dots, b\}$ such that $b = b_i + b_{i'} + w(fa(i))$ and $R^+(T_i, b_i) = R^+(T_{i'}, b_{i'}) = True$.
 - $R^-(T_{fa(i)}, b) = True$ for $b \in \{0, \dots, B\}$ if and only if there exist $b_i, b_{i'} \in \{0, \dots, b\}$ such that $b = b_i + b_{i'}$ and $R(T_i, b_i) = R(T_{i'}, b_{i'}) = True$.
 - ii. If $i, i' \in ch^-(fa(i))$ then
 - $R^+(T_{fa(i)}, b) = True$ for $b \in \{0, \dots, B\}$ if and only if there exist $b_i, b_{i'} \in \{0, \dots, b\}$ such that $b = b_i + b_{i'} + w(fa(i))$ and $R(T_i, b_i) = R(T_{i'}, b_{i'}) = True$.
 - $R^-(T_{fa(i)}, b) = True$ for $b \in \{0, \dots, B\}$ if and only if there exist $b_i, b_{i'} \in \{0, \dots, b\}$ such that $b = b_i + b_{i'}$ and $R^-(T_i, b_i) = R^-(T_{i'}, b_{i'}) = True$.
 - iii. If $i \in ch^+(fa(i))$ and $i' \in ch^-(fa(i))$ then
 - $R^+(T_{fa(i)}, b) = True$ for $b \in \{0, \dots, B\}$ if and only if there exist $b_i, b_{i'} \in \{0, \dots, b\}$ such that $b = b_i + b_{i'} + w(fa(i))$ and $R^+(T_i, b_i) = R(T_{i'}, b_{i'}) = True$.
 - $R^-(T_{fa(i)}, b) = True$ for $b \in \{0, \dots, B\}$ if and only if there exist $b_i, b_{i'} \in \{0, \dots, b\}$ such that $b = b_i + b_{i'}$ and $R(T_i, b_i) = R^-(T_{i'}, b_{i'}) = True$.
 - iv. If $i' \in ch^+(fa(i))$ and $i \in ch^-(fa(i))$ then permute i and i' and go to 1(b)iii.

2. Else, there exists a sub-graph T_i induced by V_i whose line has already been created in the tables and such that there is no other child $i' \neq i$ with the same father $fa(i)$, and the sub-tree whose root is the father $fa(i)$ (i.e. the sub-tree induced by $V_{fa(i)}$) is not created yet in the tables. Then add a new line containing the graph $T_{fa(i)}$ induced by $V_i \cup \{fa(i)\}$ and set:
- $R^-(T_{fa(i)}, b) = True$ if and only if $R^-(T_i, b) = True$ when $i \in ch^-(fa(i))$, $R(T_i, b) = True$ otherwise.
 - $R^+(T_{fa(i)}, b) = True$ if and only if $R^+(T_i, b - w(fa(i))) = True$ when $i \in ch^-(fa(i))$, $R(T_i, b - w(fa(i))) = True$ otherwise.

In case 1a, $T_{i,i'}$ is a forest and not a tree, but we use the boolean $R(T_{i,i'}, b)$ which is defined on a tree T and a nonnegative integer b . However this has no incidence on the construction of the tables R^+ and R^- . Indeed, we can transform the forest $T_{i,i'}$ into a tree $T_{ii'}$ by adding a fictitious vertex fv . If in T the arcs go from $fa(i)$ to i and i' then add in $T_{ii'}$ the arcs $(fa(i), fv)$, (fv, i) , (fv, i') , else add these arcs in the opposite direction.

The filling of each line can be realized in $O(B^2)$ because of the comparison of each value of line T_i with each value of line $T_{i'}$ in the worst case. We continue to create new lines in both tables in parallel until getting T , so we create at most $2n$ lines. Then the answer to SSG is yes if and only if $R(T, B) = True$ (i.e. if $R^-(T, B) = True$ or $R^+(T, B) = True$). In case of answer yes, the solution can be obtained by creating another table containing a feasible solution per cell valued True. In the lines containing a leaf, the solution is the empty set if the corresponding R^- is True or the leaf if the corresponding R^+ is True. Otherwise, the solution of each current True cell in the other lines is the union of the solutions of the sub-trees that compose T and we add $\{r(T)\}$ if and only if the corresponding R^+ is True and T is not the union of a subset of children of $r(T)$. ■

Remark 5. If T is an out-rooted (in-rooted resp.) tree, then it is easy to define the root $r(T)$ as the unique sink (source resp.) of T . The cases of out-rooted and in-rooted trees were previously treated in [22] where a dynamic programming algorithm was proposed. Proposition 2 is a generalization to any directed tree.

Remark 6. Proposition 2 also holds in the class of forests.

Proposition 3. MAXIMAL SSG can be solved using dynamic programming in oriented trees.

Proof. Let $I = (T, w, B)$ be an instance of MAXIMAL SSG where $T = (V, A)$ is a tree. We rename the vertices v_1, \dots, v_n of V in such a way that for all $i \in \{1, \dots, n\}$, v_i is the sink of minimum weight among the sinks of the subgraph induced by the set of vertices $V \setminus \{v_1, \dots, v_{i-1}\}$. Hence, the root is necessarily v_n . Wlog., assume $w(V) > B$, since otherwise V is an optimal solution.

Let $k \geq 1$. We denote by T_k the forest induced by $\{v_{k+1}, \dots, v_n\} \setminus asc_G(v_k)$. An example of such construction is given in Figure 7.

Let \mathcal{S} be the set of feasible solutions of MAXIMAL SSG and $\mathcal{S}_k \subseteq \mathcal{S}$ be the subset satisfying:

$$\forall S \in \mathcal{S}_k, \quad \{v_1, \dots, v_{k-1}\} \subseteq S \text{ and } v_k \notin S \quad (8)$$

for $k \in \{1, \dots, n\}$ where \mathcal{S}_1 is reduced to subsets S such that $v_1 \notin S$. It is obvious that $\{\mathcal{S}_1, \dots, \mathcal{S}_n\}$ is a partition of \mathcal{S} where some parts may be empty.

Let $k \geq 1$ and let $I_k = (T_k, w, B_k)$ be an instance of SSG where T_k is the forest defined above and $B_k = B - \sum_{i=1}^{k-1} w(v_i)$ where $B_1 = B$. Then we show there exists $S \in \mathcal{S}_k$ if and only if $R(T_k, b) = True$ for some $b \in (B_{k+1}, B_k]$ where $R(T_k, b)$ is the boolean table defined in the proof of Proposition 2, i.e. the instance I_k admits a feasible solution for SSG with weight $b \in (B_{k+1}, B_k]$.

First, assume there exists $S \in \mathcal{S}_k$. Then

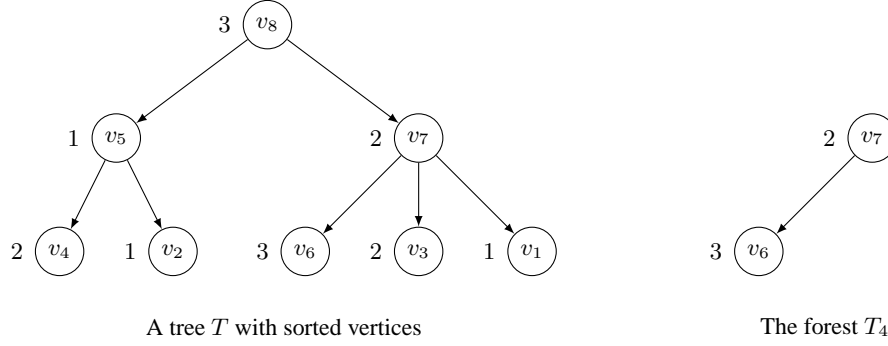


Fig. 7. An example of construction of the forest T_k for $k = 4$.

- the restriction S_k of S to T_k satisfies (1);
- using (8), we know that $v_k \notin S$, so using Property 1, $S \setminus \{v_1, \dots, v_{k-1}\} = S_k$. Using $w(S) \leq B$, we conclude that $w(S_k) \leq B_k$;
- since $S \in \mathcal{S}_k$ and T is a tree (so, a DAG), it follows that S satisfies (6). But $v_k \notin S$ and $S \cup \{v_k\}$ satisfies (1), so it must be $w(S) + w(v_k) > B$. Thus, $w(S_k) > B_k - w(v_k) = B_{k+1}$.

Hence, we may conclude that $R(T_k, w(S_k)) = \text{True}$ and $B_{k+1} < w(S_k) \leq B_k$.

Conversely, assume that there exists $b \in (B_{k+1}, B_k]$ such that $R(T_k, b) = \text{True}$. Then there exists $S_k \subseteq T_k$ such that S_k satisfies (1) and $w(S_k) = b \leq B_k$. In the initial graph T , we have

- $S = S_k \cup \{v_1, \dots, v_{k-1}\}$ satisfies (1),
- $w(S) = w(S_k) + \sum_{i=1}^{k-1} w(v_i) \leq B$,
- Let us prove that S satisfies the maximality constraints (6). Using Remark 2, let v be a sink of $G - S$; so $v = v_i$ for $i \geq k$. We have $w(S) + w(v) \geq w(S) + w(v_k) = b + \sum_{i=1}^{k-1} w(v_i) + w(v_k) > B_{k+1} + \sum_{i=1}^{k-1} w(v_i) + w(v_k) = B_k + \sum_{i=1}^{k-1} w(v_i) = B$.

We conclude that $S = S_k \cup \{v_1, \dots, v_{k-1}\} \in \mathcal{S}_k$.

Then, the value of the optimal solution of MAXIMAL SSG is $\min_{k \leq n} b_k$ with $b_k = \min\{b \in (B_{k+1}, B_k] \text{ for which } R(T_k, b) = \text{True}\}$. ■

Proposition 4. SSGW can be solved using dynamic programming in in-rooted and out-rooted trees.

Proof. In in-rooted trees, the weak digraph constraints (4) are equivalent to the digraph constraints (1). Hence, SSGW is equivalent to solve SSG and the result holds by Proposition 2.

Let us now consider the case of out-rooted trees. Let $P(T, b)$ be a boolean defined on an out-rooted tree and an integer $b \in \{0, \dots, B\}$ such that $P(T, b) = \text{True}$ if and only if there exists $S \subseteq V$ satisfying the weak digraph constraints (4) with $w(S) = b$. Let $r(T)$ be the anti-root of T . Then it is easy to see that $P(T, b)$ is defined recursively as follows.

$$P(T, b) = \begin{cases} \text{False, if } b > w(V), \\ \text{True, if } b = w(V) \wedge \forall i \in \text{ch}(r(T)) : P(T_i, w(V_i)) = \text{True}, \\ \text{True, if } \exists I \subseteq \text{ch}(r(T)) : \forall i \in I : P(T_i, b_i) = \text{True for some } b_i \in \{0, \dots, b\} \\ \quad \text{such that } \sum_{i \in I} b_i = b \vee \sum_{i \in I} b_i = b - w(r(T)), \\ \text{False, else.} \end{cases}$$

■

The generalization of Proposition 4 to any directed tree is an open problem.

Note that the dynamic programming algorithms presented in Propositions 2, 3 and 4 can be used to deduce fully polynomial time approximation schemes.

7 Approximation schemes for SSG and MAXIMAL SSG in DAG

An instance is a graph $G = (V, A)$, a bound B and a nonnegative weight $w(v)$ for each node $v \in V$. Using Lemma 2, we can suppose that G is a connected DAG. Being a DAG is a hereditary property, so every nonempty subgraph of G possesses a source and a sink. We propose two polynomial approximation schemes for SSG and MAXIMAL SSG, respectively. They both consist in building a partial solution with an exhaustive search of k nodes (k is a part of the input) and complete it in a greedy manner. In both cases, the time complexity of the algorithm is dominated by the first phase which requires $O(|V|^k |A|)$ elementary operations.

7.1 SSG in DAG

Algorithm 1 consists in building every possible subset S of V such that $|S| \leq k$. The nodes of $\text{desc}_G(S)$ are put in a partial solution Sol_S if $w(\text{desc}_G(S)) \leq B$. Next, the nodes of $V \setminus (\text{asc}_G(\kappa(S)) \cup \text{desc}_G(S))$ are considered by nonincreasing marginal contribution (that is $w(\text{desc}_G(z) \setminus \text{Sol}_S) = w(\text{desc}_{G'}(z))$) and $\text{desc}_{G'}(z)$ is added to Sol_S if the budget B is not exceeded, until a feasible solution is obtained. The algorithm finally outputs the best solution Sol^* that was constructed (the one of maximum weight).

Theorem 4. *Algorithm 1 is a PTAS for SSG in DAG.*

Proof. Let O be an optimal solution to SSG. Let q denote $|\kappa(O)|$ and suppose $\kappa(O) = \{v_1, v_2, \dots, v_q\}$. If $k \geq q$, then the algorithm finds $\kappa(O)$ and deduces O . Henceforth, $k + 1 \leq q$. Wlog., assume that the nodes of $\kappa(O)$ have been sorted according to their marginal contribution; so, let $\hat{w}(v_i)$ denote $w(\text{desc}_G(v_i) \setminus \text{desc}_G(\{v_1, \dots, v_{i-1}\}))$ and $\hat{w}(v_i) \geq \hat{w}(v_{i+1})$, $\forall i \in [1..q - 1]$. The value of the optimal solution $w(O)$ is equal to $\sum_{i=1}^q \hat{w}(v_i)$. Let i^* be such that $\hat{w}(v_i) \leq \frac{B}{k+1}$ iff $i \geq i^*$.

If $i^* > k + 1$, then $\hat{w}(v_1) \geq \dots \geq \hat{w}(v_{k+1}) > \frac{B}{k+1}$ and $w(O) > B$, contradiction with (2). Thus, $i^* \leq k + 1 \leq q$ and the algorithm can guess $\{v_1, \dots, v_{i^*-1}\}$ during its first phase. Henceforth, we analyze the iteration of the algorithm where $S = \{v_1, \dots, v_{i^*-1}\}$. Of course, $w(\text{Sol}^*) \geq w(\text{Sol}_S)$.

During the second phase, if the algorithm inserts $\{v_{i^*}, v_{i^*+1}, \dots, v_q\}$, then Sol_S is clearly optimal. Otherwise, let j be the smallest element of $[i^*..q]$ such that $v_j \notin \text{Sol}_S$. We cannot add v_j to Sol_S because $w(\text{Sol}_S) > B - w(\text{desc}_{G'}(v_j)) = B - w(\text{desc}_G(v_j) \setminus \text{desc}_G(\text{Sol}_S))$. Since $w(\text{desc}_G(v_j) \setminus \text{desc}_G(\text{Sol}_S)) \leq w(\text{desc}_G(v_j) \setminus \text{desc}_G(\{v_1, \dots, v_{j-1}\})) = \hat{w}(v_j)$, we get that $w(\text{Sol}_S) > B - \frac{B}{k+1} \geq \frac{k}{k+1} w(O)$. ■

Algorithm 1:

Data: a DAG $G = (V, A)$, B , w and k
Result: a set of nodes Sol^* satisfying (1) and (2)

```

1  $Sol^* \leftarrow \emptyset$ 
2 for all  $S \subseteq V$  such that  $|S| \leq k$  do
3   if  $w(desc_G(S)) \leq B$  then
4      $V' \leftarrow V \setminus (asc_G(\kappa(S)) \cup desc_G(S))$ 
5      $G' \leftarrow G[V']$ 
6      $Sol_S \leftarrow desc_G(S)$ 
7     while  $V' \neq \emptyset$  do
8       Choose a source node  $z$  of  $G'$ , maximizing  $w(desc_{G'}(z))$ 
9       if  $w(Sol_S) + w(desc_{G'}(z)) \leq B$  then
10          $Sol_S \leftarrow Sol_S \cup desc_{G'}(z)$ 
11          $V' \leftarrow V' \setminus desc_{G'}(z)$ 
12       else
13          $V' \leftarrow V' \setminus \{z\}$ 
14          $G' \leftarrow G[V']$ 
15     if  $w(Sol_S) > w(Sol^*)$  then
16        $Sol^* \leftarrow Sol_S$ 
17 return  $Sol^*$ 

```

7.2 MAXIMAL SSG in DAG

By Lemma 3, we know that constraint (3) can be replaced by (6) in DAG. Algorithm 2 consists in building every possible set $S \subseteq V$ such that $|S| \leq k$. The nodes of $desc_G(S)$ are put in a partial solution Sol_S if $w(desc_G(S)) \leq B$. Next, the sinks of $G[V \setminus (Sol_S)]$ are considered by nondecreasing weight and greedily added to Sol_S if the budget is not exceeded, until a feasible solution is obtained. The algorithm finally outputs the best solution Sol^* that was constructed (the one of minimum weight).

Theorem 5. *Algorithm 2 is a PTAS for MAXIMAL SSG in DAG.*

Proof. Let O be an optimal solution to MAXIMAL SSG. Let q denote $|\kappa(O)|$ and suppose $\kappa(O) = \{v_1, v_2, \dots, v_q\}$. If $k \geq q$, then the algorithm finds $\kappa(O)$ and deduces O . Henceforth, $k + 1 \leq q$. As previously, assume that the nodes of $\kappa(O)$ have been sorted according to their marginal contribution. Let $\hat{w}(v_i)$ denote $w(desc_G(v_i) \setminus desc_G(\{v_1, \dots, v_{i-1}\}))$, and suppose that $\hat{w}(v_i) \geq \hat{w}(v_{i+1})$, $\forall i \in \{1, \dots, q-1\}$. The value of the optimal solution $w(O)$ is equal to $\sum_{i=1}^q \hat{w}(v_i)$. Observe that for every $j \in [1..q-1]$ and every node $x \in O \setminus desc_G(\{v_1, \dots, v_j\})$, it holds that:

$$w(x) \leq \hat{w}(v_{j+1}) \leq w(desc_G(\{v_1, \dots, v_j\}))/j \leq w(O)/j \quad (9)$$

Consider the iteration of the algorithm where $S = \{v_1, \dots, v_k\}$. The corresponding solution Sol_S consists of $desc_G(\{v_1, \dots, v_k\})$ plus some other nodes that are subsequently added in a greedy manner. Let z_i denote the i -th node inserted during the greedy phase. Let s be smallest index such that $z_s \notin O$ (if z_s does not exist, then Sol_S must be optimal).

Note that $O \setminus (desc_G(\{v_1, \dots, v_k\}) \cup \{z_1, \dots, z_{s-1}\}) \neq \emptyset$, otherwise z_s can be added to O , violating (3). Let u be a sink of $G[V \setminus (desc_G(\{v_1, \dots, v_k\}) \cup \{z_1, \dots, z_{s-1}\})]$, such that $u \in O$.

Algorithm 2:

Data: a DAG $G = (V, A)$, B , w and k
Result: a set of nodes Sol^* satisfying (1), (2) and (6)

```

1  $Sol^* \leftarrow V$ 
2 for all  $S \subseteq V$  such that  $|S| \leq k$  do
3   if  $w(desc_G(S)) \leq B$  then
4      $V' \leftarrow V \setminus (desc_G(S))$ 
5      $G' \leftarrow G[V']$ 
6      $Sol_S \leftarrow desc_G(S)$ 
7     while  $Sol_S$  does not satisfy (6) do
8       Within the sinks of  $G'$ , choose one, say  $z$ , of minimum weight
9        $Sol_S \leftarrow Sol_S \cup \{z\}$ 
10       $V' \leftarrow V' \setminus \{z\}$ 
11       $G' \leftarrow G[V']$ 
12      if  $w(Sol_S) < w(Sol^*)$  then
13         $Sol^* \leftarrow Sol_S$ 
14 return  $Sol^*$ 

```

Such a vertex exists because $G[O]$ is a DAG and $O \supset (\{v_1, \dots, v_k\}) \cup \{z_1, \dots, z_{s-1}\}$. Using (9), we know that $w(u) \leq w(O)/k$. The greedy phase of the algorithm consists in adding to the current solution a sink of minimum weight, so $w(u) \geq w(z_s)$. Because O is feasible and $O \supset (\{v_1, \dots, v_k\}) \cup \{z_1, \dots, z_{s-1}\}$, $O \cup \{z_s\}$ must violate the budget constraint, i.e. $w(O) + w(z_s) > B$. Sol_S satisfies the budget constraint. We deduce that $w(Sol^*) \leq w(Sol_S) \leq B < w(O) + w(z_s) \leq w(O) + w(u) \leq \frac{k+1}{k}w(O)$. ■

Remark 7. Note that for $k = 0$, Algorithm 1 and 2 are greedy algorithms and it is not difficult to prove that their exact approximation bounds are $1/2$ and 2 for SSG and MAXIMAL SSG, respectively.

8 Conclusion and perspectives

We presented in this article some complexity results for the problem of (MAXIMAL) SUBSET SUM WITH (WEAK) DIGRAPH CONSTRAINTS. We designed complexity results according to the class of the input digraph, namely regular graphs (for SSG), directed acyclic graphs and oriented trees. It would be interesting to see the tightness of the complexity results in these classes. This was done only for SSG in regular graphs.

References

1. P. Alimonti and V. Kann. Some APX-completeness results for cubic graphs. *Theor. Comput. Sci.*, 237(1-2):123–134, 2000.
2. E. M. Arkin, M. A. Bender, J. S. B. Mitchell, and S. Skiena. The lazy bureaucrat scheduling problem. *Information ans Computation*, 184(1):129–146, 2003.
3. J. Bang-Jensen and P. Hell. Fast algorithms for finding hamiltonian paths and cycles in in-tournament digraphs. *Discrete applied mathematics*, 41(1):75–79, 1993.
4. R. I. Becker and Y. Perl. The shifting algorithm technique for the partitioning of trees. *Discrete Applied Mathematics*, 62(1-3):15–34, 1995.

5. S. Bervoets, V. Merlin, and G. J. Woeginger. Vote trading and subset sums. *Operations Research Letters*, 43:99–102, 2015.
6. N. L. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory 1736-1936*. Clarendon Press, Oxford, 1976.
7. N. Boland, A. Bley, C. Fricke, G. Froyland, and R. Sotirov. Clique-based facets for the precedence constrained knapsack problem. *Math. Program.*, 133(1-2):481–511, 2012.
8. G. Borradaile, B. Heeringa, and G. T. Wilfong. The knapsack problem with neighbour constraints. *J. Discrete Algorithms*, 16:224–235, 2012.
9. M. Chlebík and J. Chlebíková. Approximation hardness of dominating set problems in bounded degree graphs. *Inf. Comput.*, 206(11):1264–1275, 2008.
10. G. Cho and D. X. Shaw. A depth-first dynamic programming algorithm for the tree knapsack problem. *INFORMS Journal on Computing*, 9(4):431–438, 1997.
11. M. Cieliebak, S. Eidenbenz, and A. Pagourtzis. Composing equipotent teams. In *Fundamentals of Computation Theory, 14th International Symposium, FCT 2003, Malmö, Sweden, August 12-15, 2003, Proceedings*, pages 98–108, 2003.
12. M. Cieliebak, S. Eidenbenz, A. Pagourtzis, and K. Schlude. On the complexity of variations of equal sum subsets. *Nord. J. Comput.*, 14(3):151–172, 2008.
13. C. Eggermont and G. J. Woeginger. Motion planning with pulley, rope, and baskets. *Theory Comput. Syst.*, 53(4):569–582, 2013.
14. B. Esfahbod, M. Ghodsi, and A. Sharifi. Common-deadline lazy bureaucrat scheduling problems. In F. K. H. A. Dehne, J. Sack, and M. H. M. Smid, editors, *Algorithms and Data Structures, 8th International Workshop, WADS 2003, Ottawa, Ontario, Canada, July 30 - August 1, 2003, Proceedings*, volume 2748 of *Lecture Notes in Computer Science*, pages 59–66. Springer, 2003.
15. L. Gai and G. Zhang. On lazy bureaucrat scheduling with common deadlines. *Journal of Combinatorial Optimization*, 15(2):191–199, 2008.
16. M. Garey and D. Johnson. *Computers and intractability*, volume 174. Freeman New York, 1979.
17. L. Gourvès, J. Monnot, and A. Pagourtzis. The lazy matroid problem. In J. Diaz, I. Lanese, and D. Sangiorgi, editors, *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*, volume 8705 of *Lecture Notes in Computer Science*, pages 66–77. Springer, 2014.
18. L. Gourvès, J. Monnot, and A. T. Pagourtzis. The lazy bureaucrat problem with common arrivals and deadlines: approximation and mechanism design. In *Fundamentals of Computation Theory*, pages 171–182. Springer, 2013.
19. M. T. Hajiaghayi, K. Jain, L. C. Lau, I. I. Mandoiu, A. Russell, and V. V. Vazirani. Minimum multicolored subgraph problem in multiplex PCR primer set selection and population haplotyping. In V. N. Alexandrov, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, editors, *Computational Science - ICCS 2006, 6th International Conference, Reading, UK, May 28-31, 2006, Proceedings, Part II*, volume 3992 of *Lecture Notes in Computer Science*, pages 758–766. Springer, 2006.
20. M. M. Halldórsson. Approximating the minimum maximal independence number. *Inf. Process. Lett.*, 46(4):169–172, 1993.
21. J. Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 627–636. IEEE, 1996.
22. D. S. Johnson and K. A. Niemi. On knapsacks, partitions, and a new dynamic programming technique for trees. *Mathematics of Operations Research*, 8(1):1–14, 1983.
23. H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.
24. S. G. Kolliopoulos and G. Steiner. Partially ordered knapsack and applications to scheduling. *Discrete Applied Mathematics*, 155(8):889–897, 2007.
25. A. Kothari, S. Suri, and Y. Zhou. Interval subset sum and uniform-price auction clearing. In L. Wang, editor, *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*, volume 3595 of *Lecture Notes in Computer Science*, pages 608–620. Springer, 2005.
26. D. Manlove. On the algorithmic complexity of twelve covering and independence parameters of graphs. *Discrete Applied Mathematics*, 91(1-3):155–175, 1999.
27. G. J. Woeginger and Z. Yu. On the equal-subset-sum problem. *Inf. Process. Lett.*, 42(6):299–302, 1992.